Computer Science at Oxford

Michael Spivey

Tutor, Oriel College

Department of
COMPUTER
SCIENCE

Computer Science at Oxford

- What's Computer Science about?
- The Oxford courses
- Four myths about Oxford

Solving Sudoku by computer

That program uses a sensible approach:

- systematically explore all possibilities.

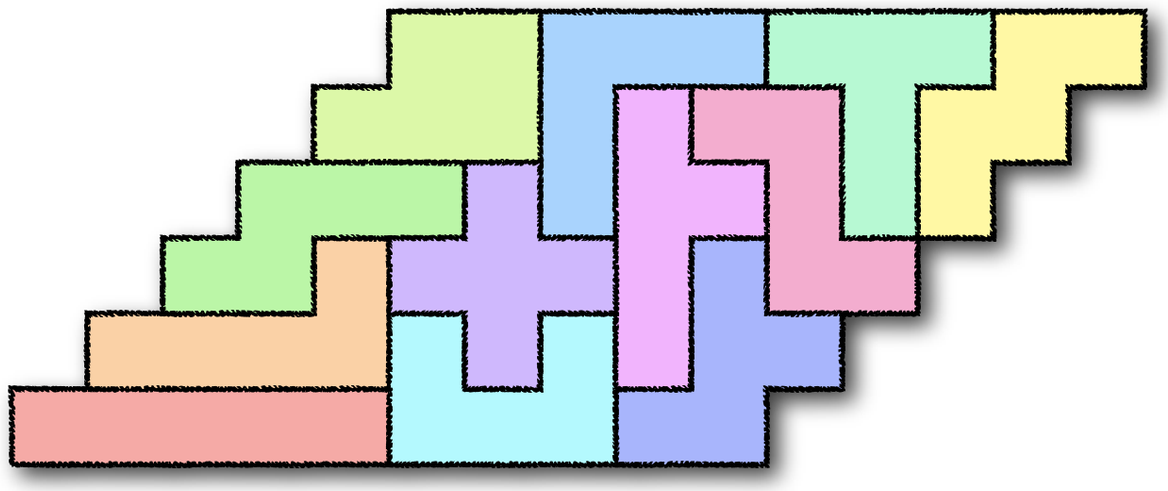
But ...

- it's using a stupid strategy.
- it's solving the wrong problem.

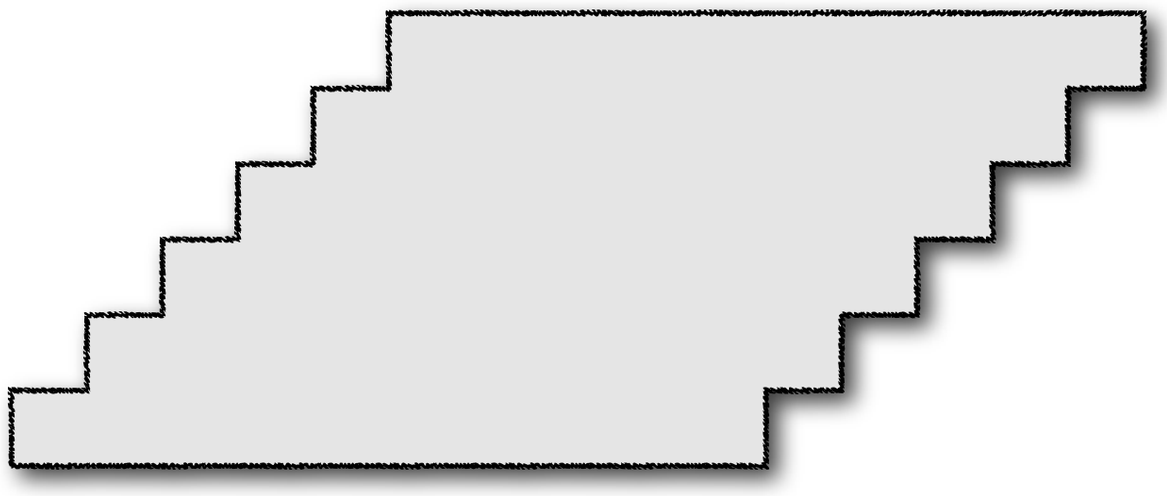
Pentominoes

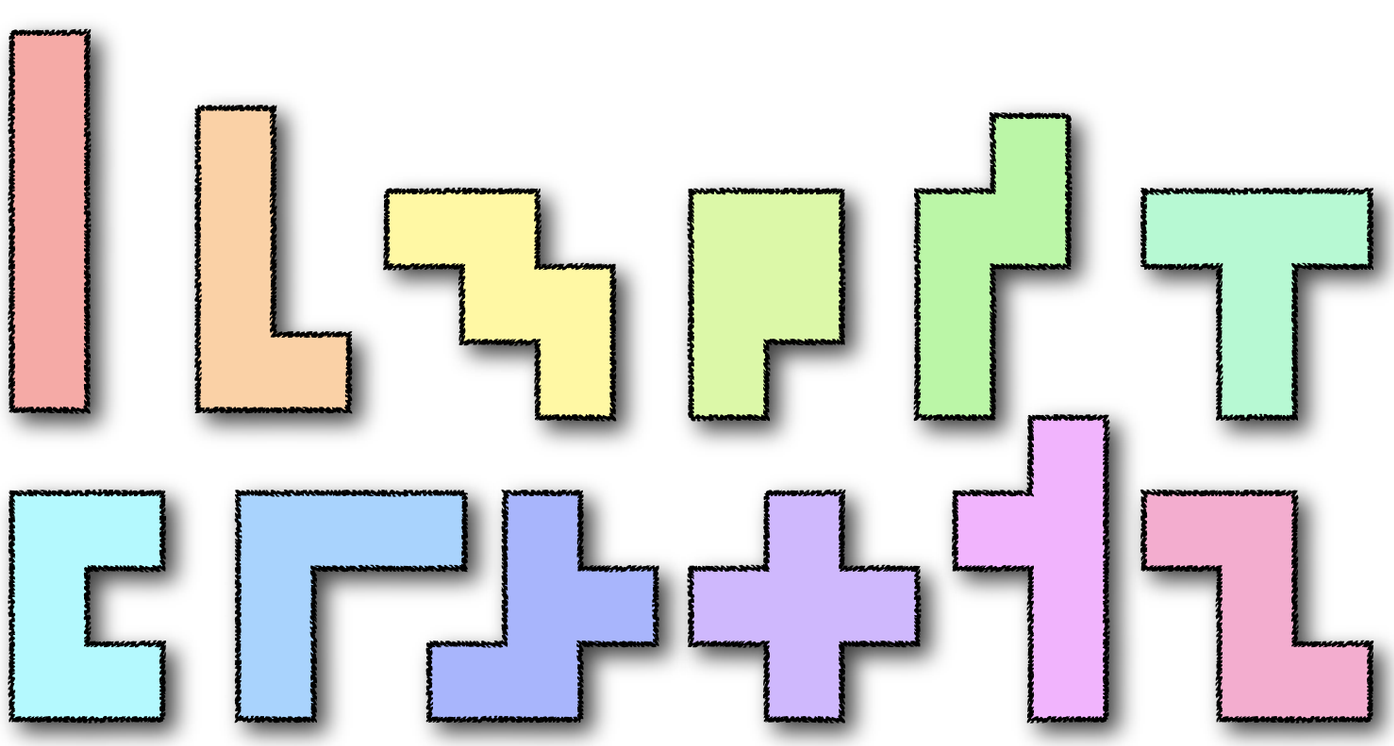
Fill a specified shape

- by covering each square *exactly once*,
- and using each piece *exactly once.*

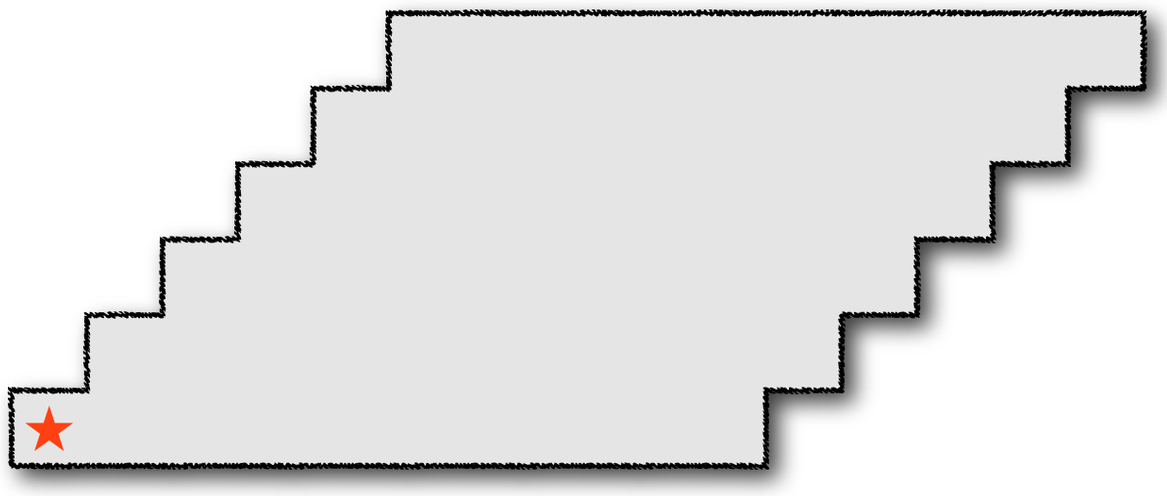


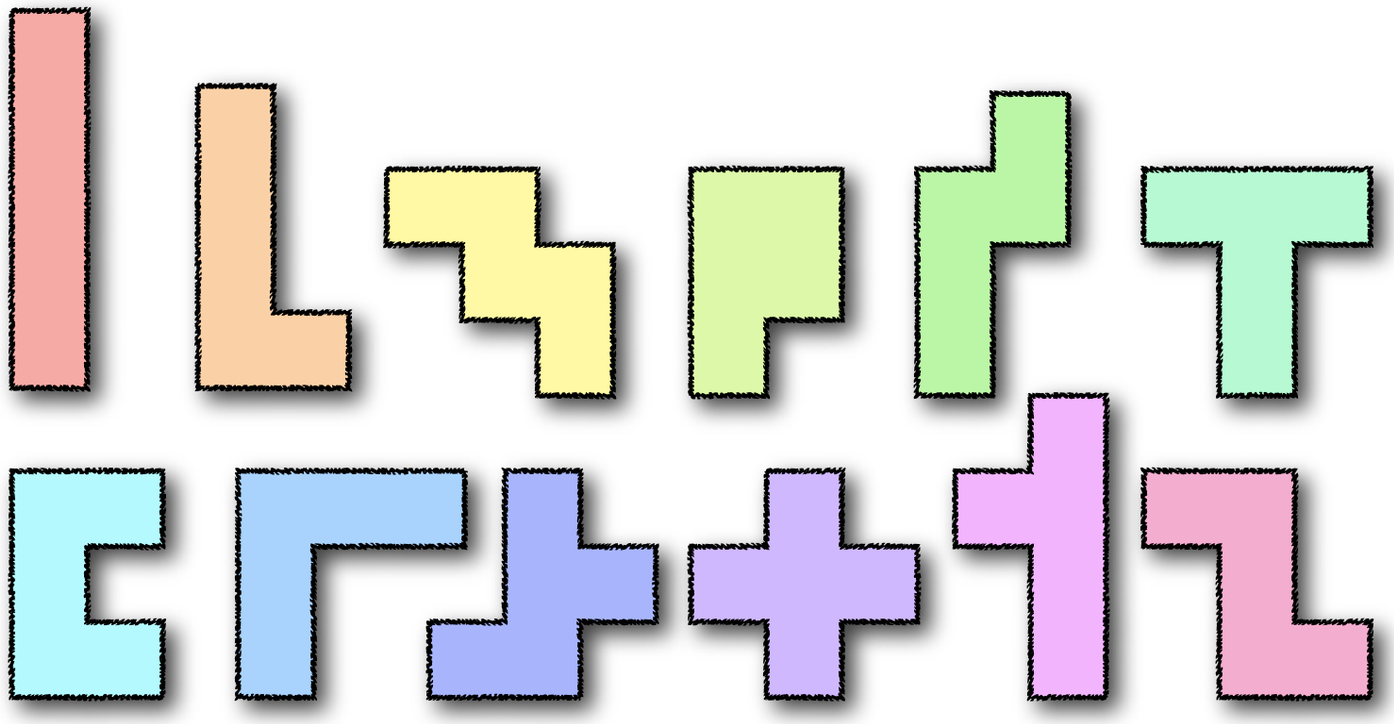
Start with an empty board



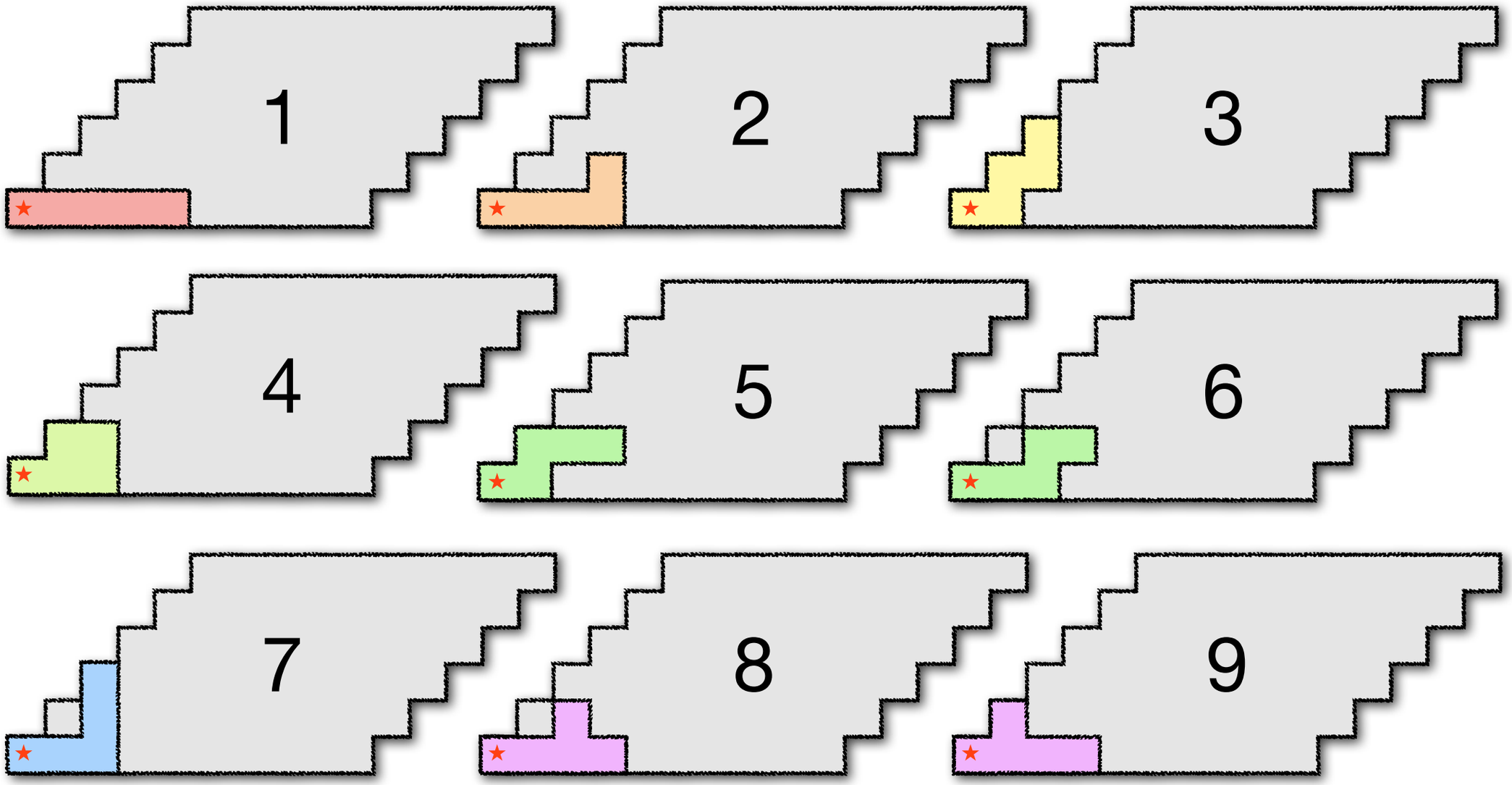


Focus on one square

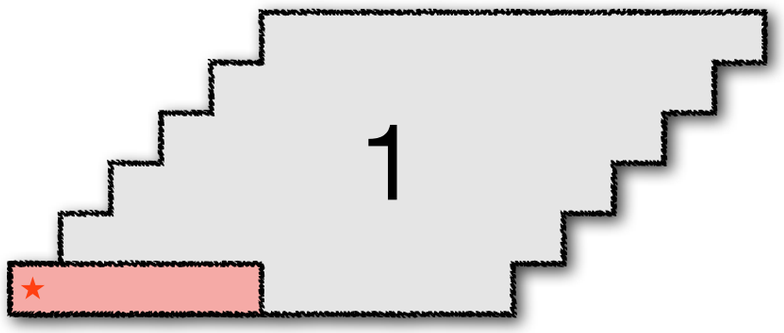


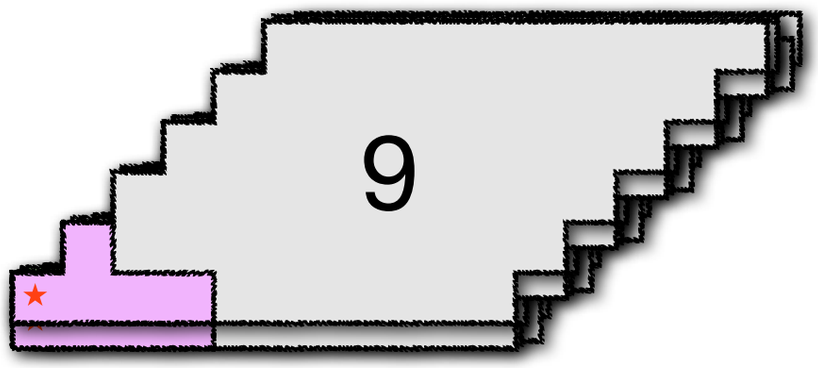


Nine ways to cover it

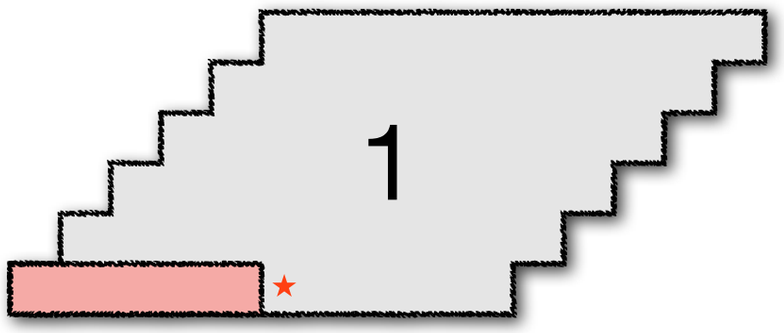


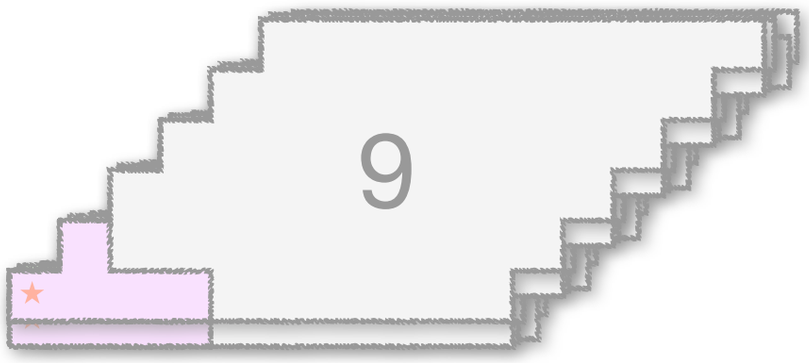
Put eight of them aside



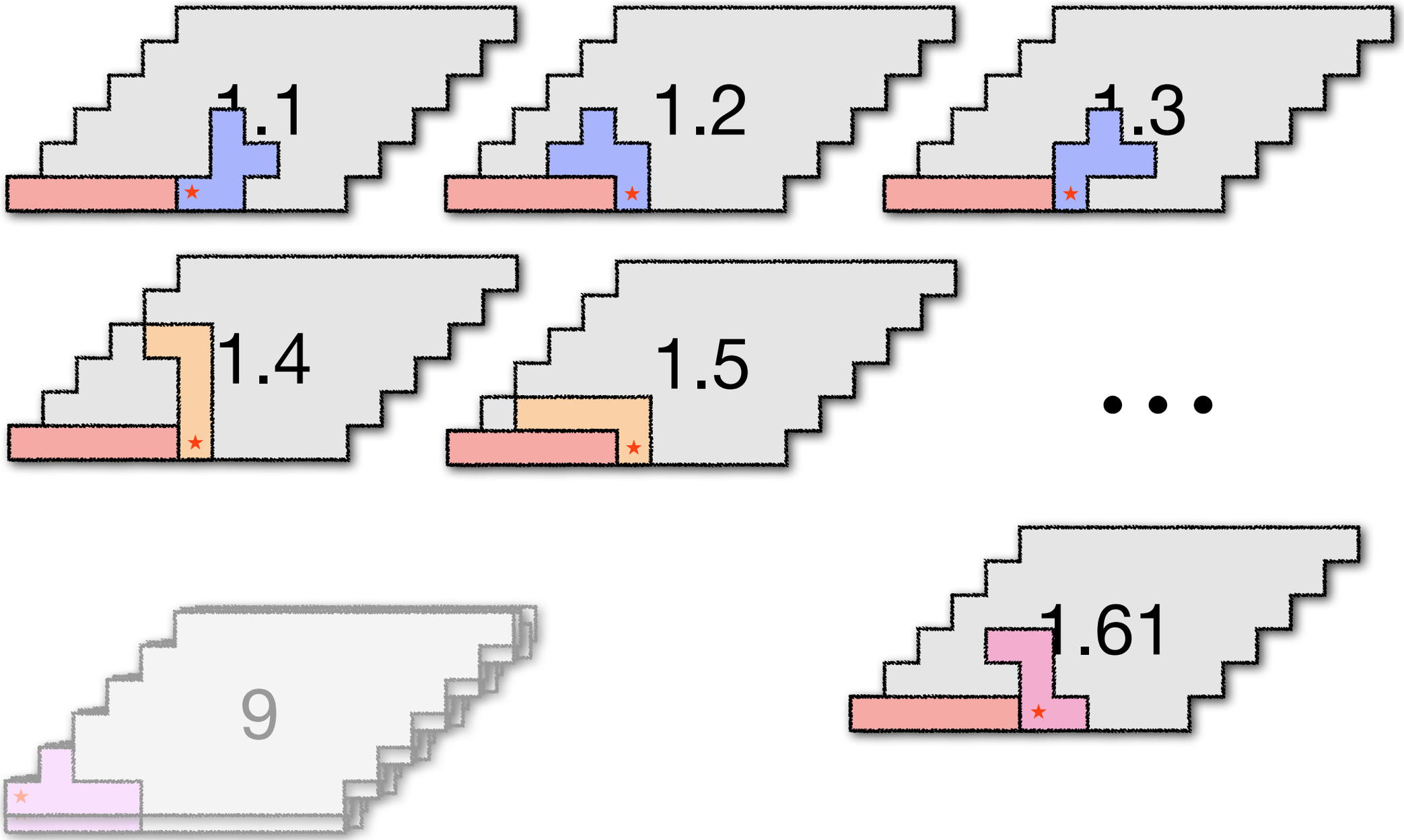


Focus on another square

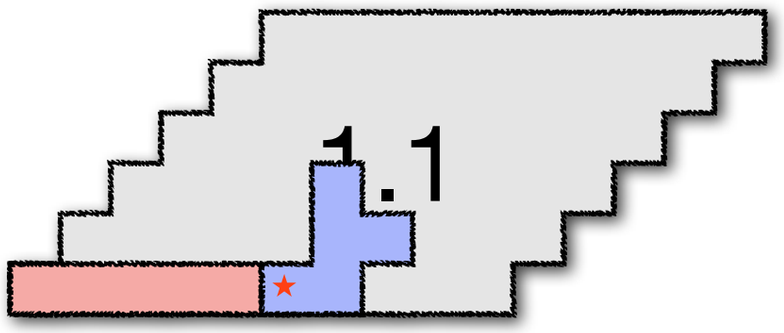


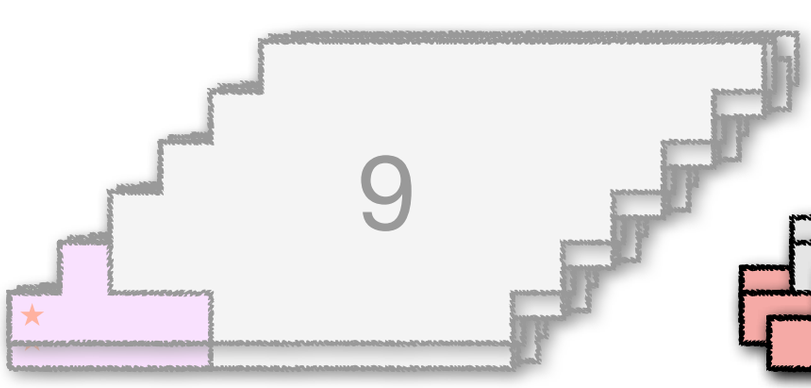


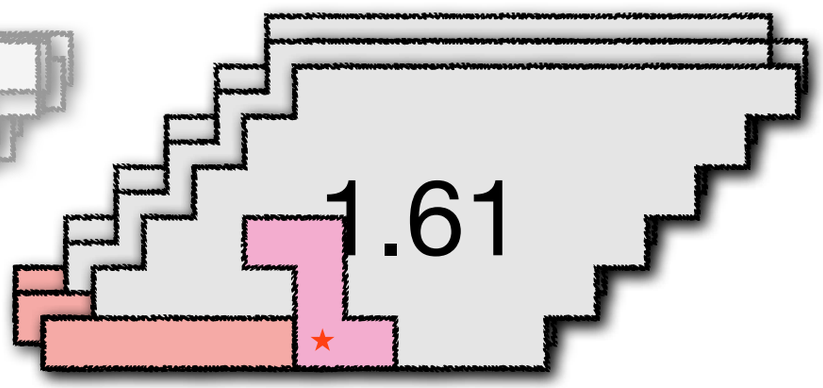
61 ways to cover it



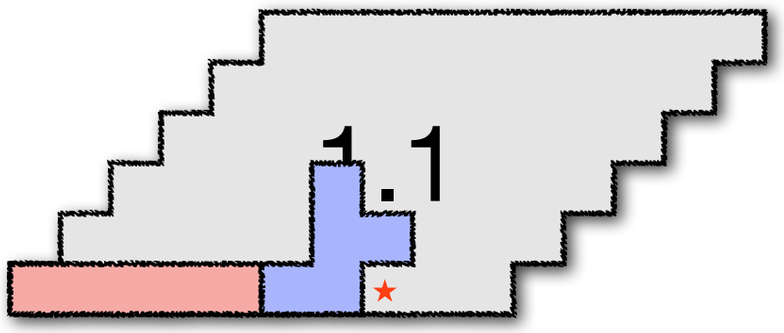
Put 60 aside …

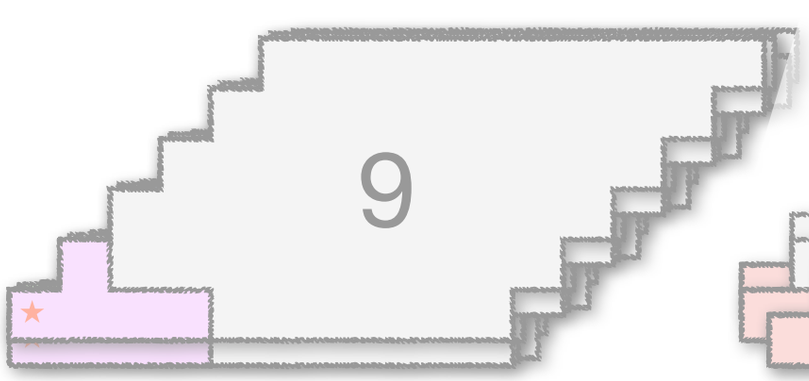


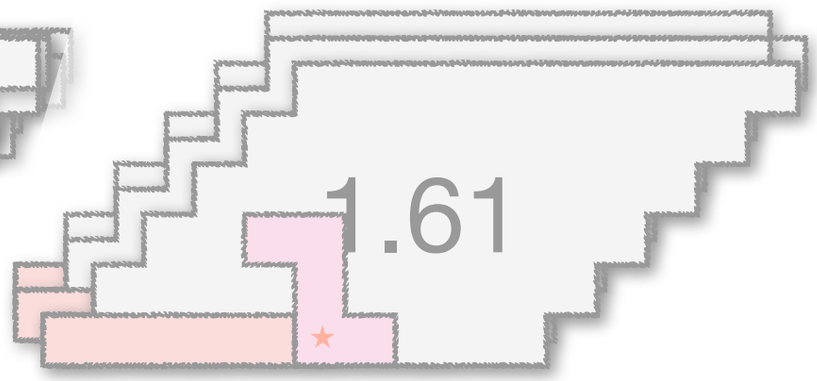




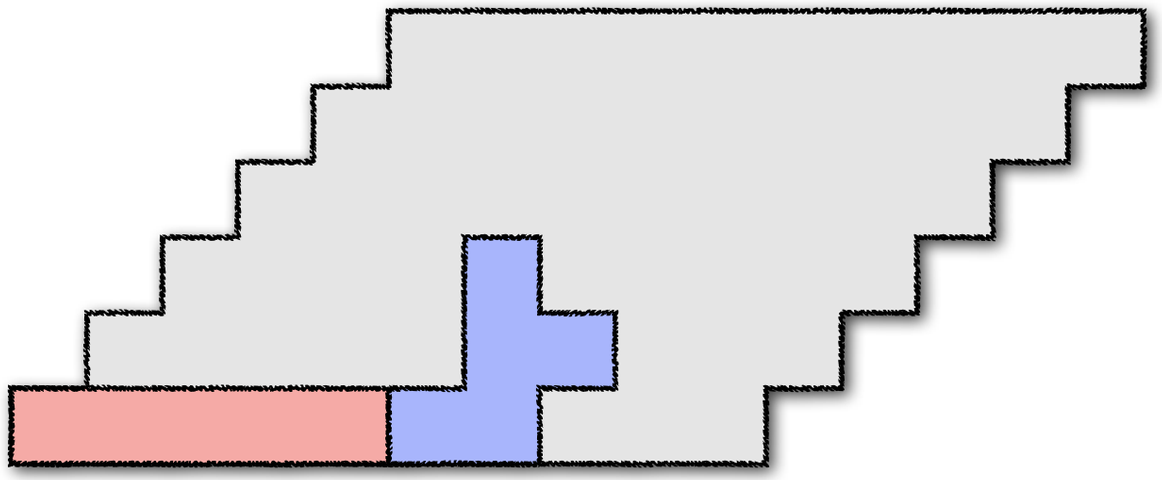
Focus on another square



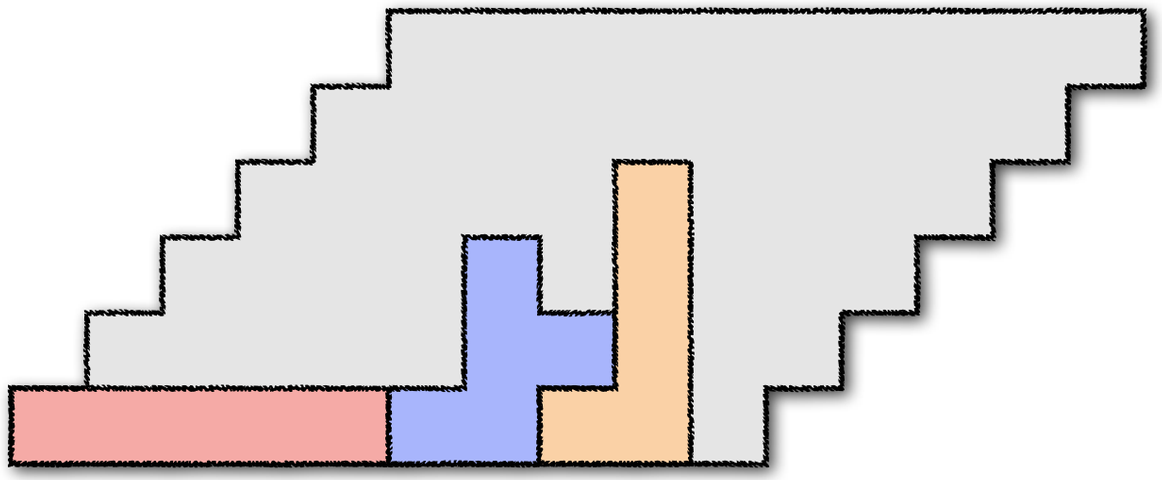




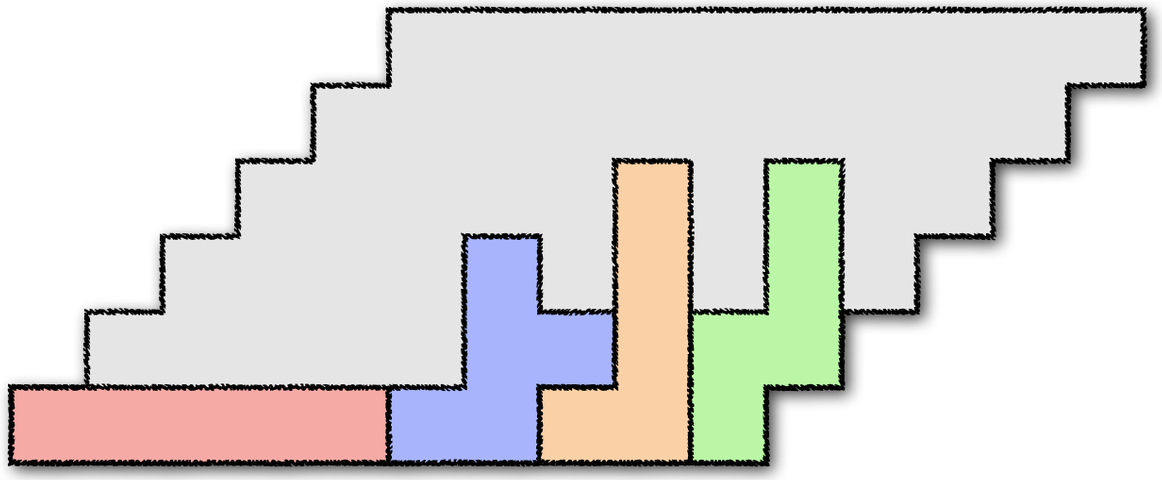
… and so it goes on



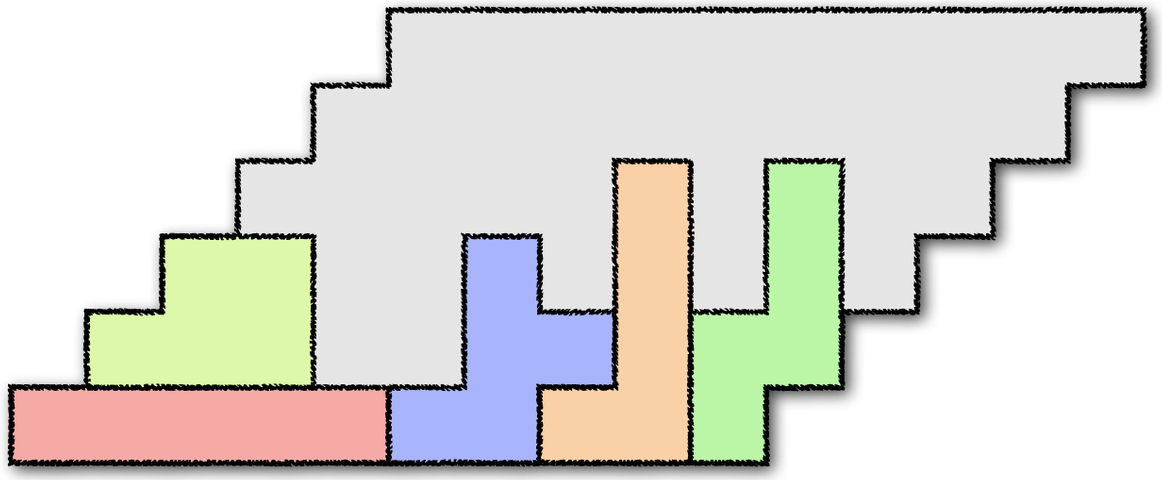
… and so it goes on



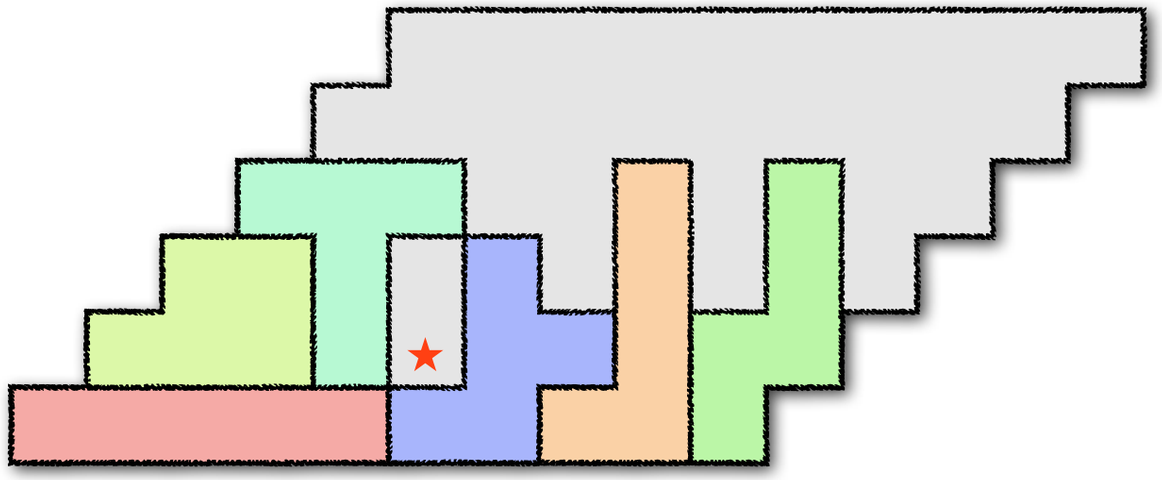
… and so it goes on



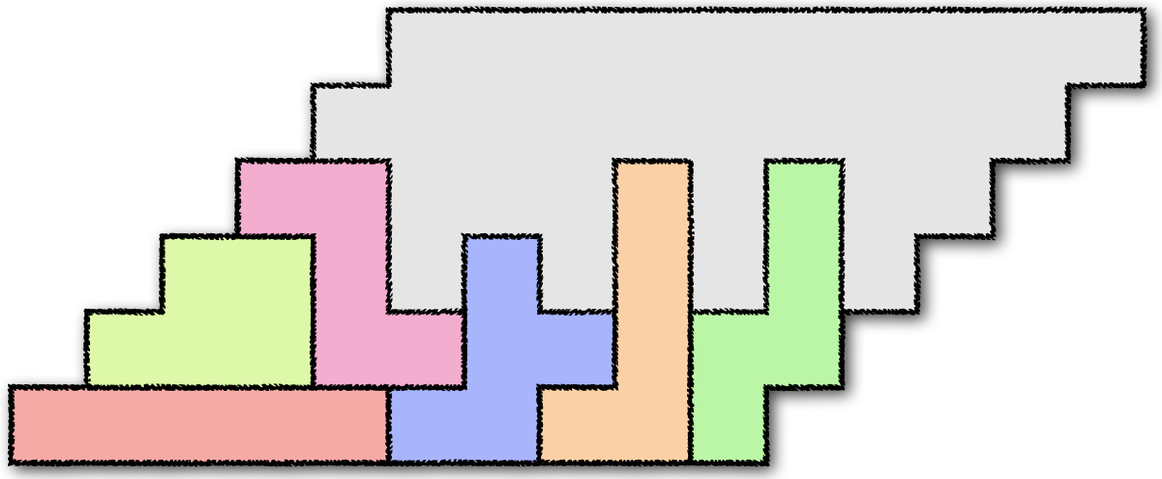
… and so it goes on



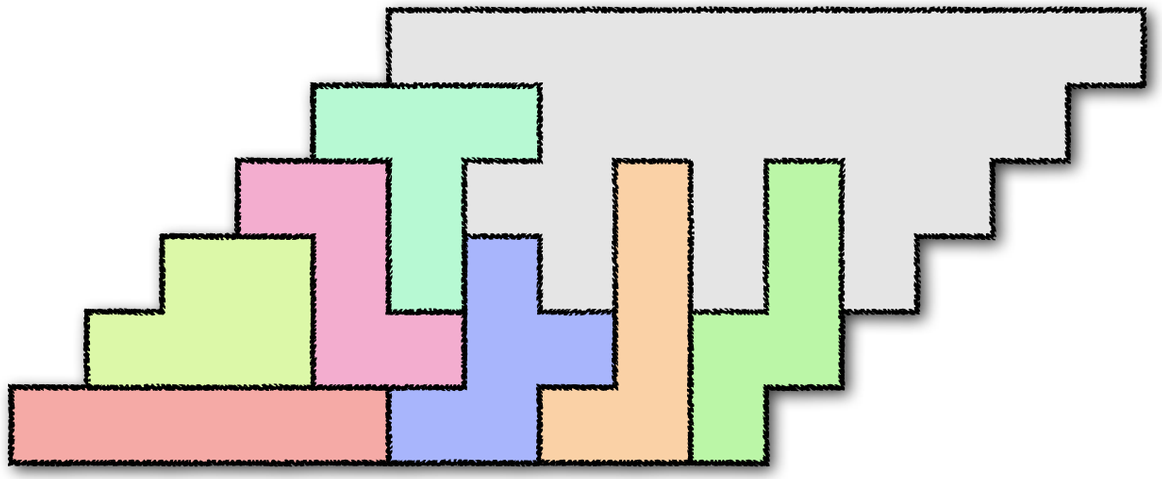
A dead end!



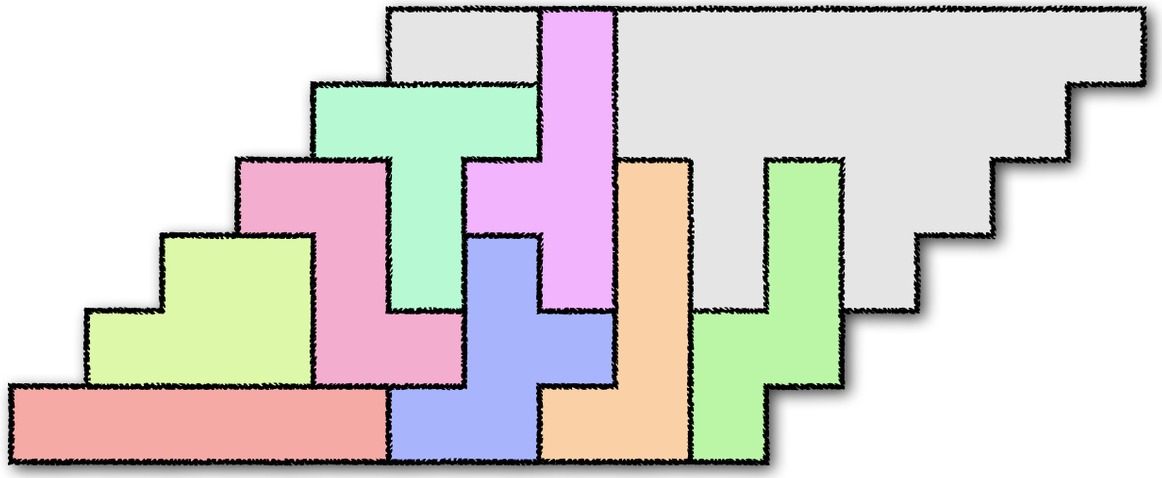
Backtrack and try again



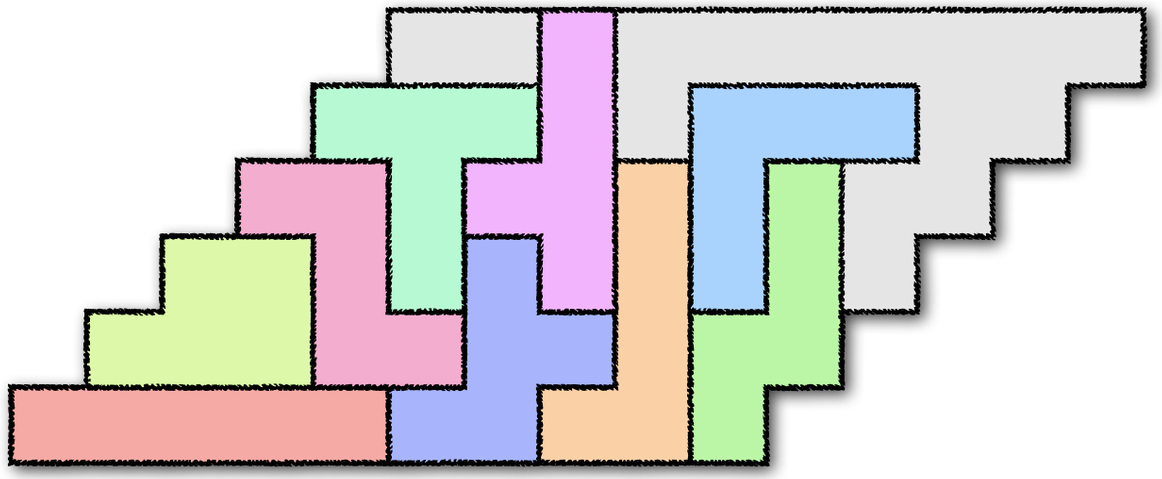
Backtrack and try again



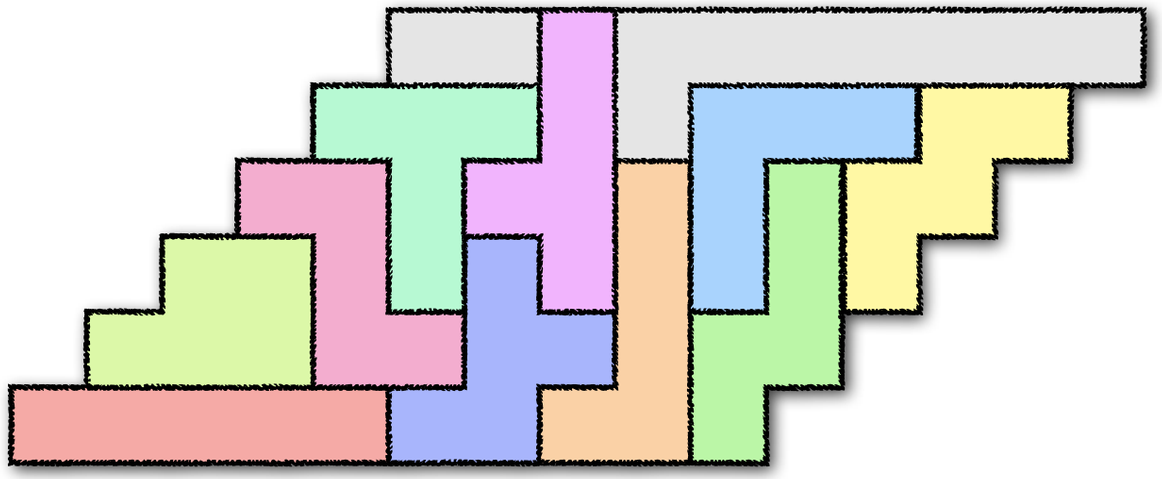
A hopeless case



A hopeless case



A hopeless case



Backtracking search

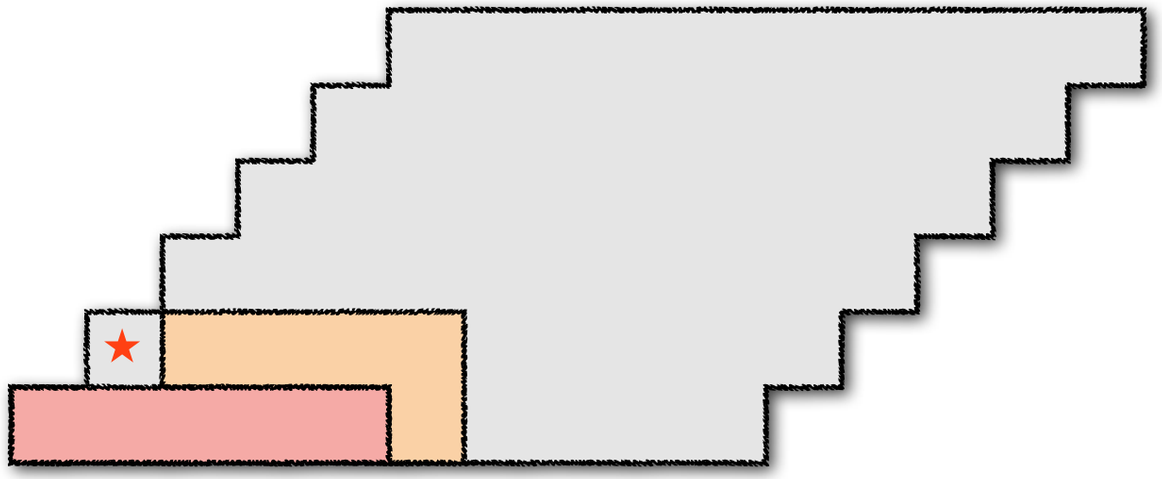
If the puzzle isn't complete:

1. focus on a square (or a tile).
2. generate all ways of covering it (or placing it).
3. explore each of them in turn.

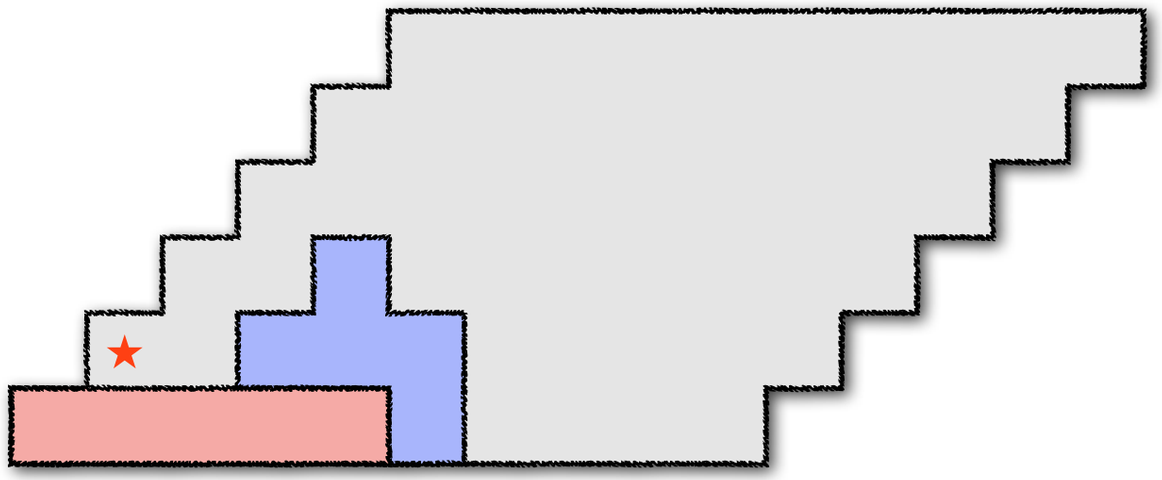
When you reach a dead end, go back to the most recent pile of unexplored states.

A naturally *recursive* process.

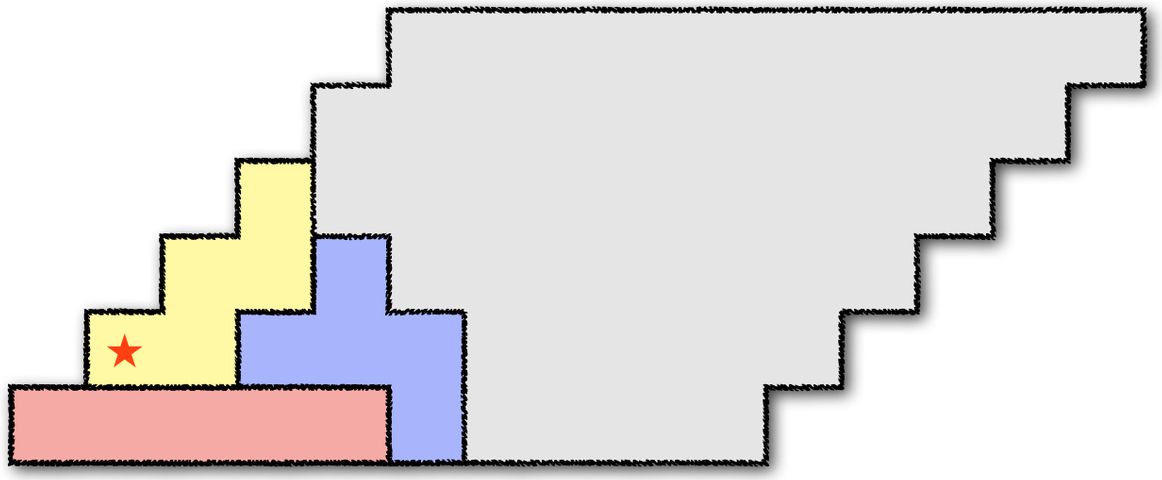
A hopeless case



A forced move



A forced move



Focussing wisely

There's a big advantage in focussing on a square with *few* possibilities:

- if zero, a hopeless case.
- if one, a forced move.
- if more than one, the fewer the better for a smaller search space.

Another kind of puzzle

Given a rectangular array ...

	A	B	C	D	E	F	G
1			*		*	*	
2	*			*			*
3		*	*			*	
4	*			*			
5		*					*
6				*	*		*

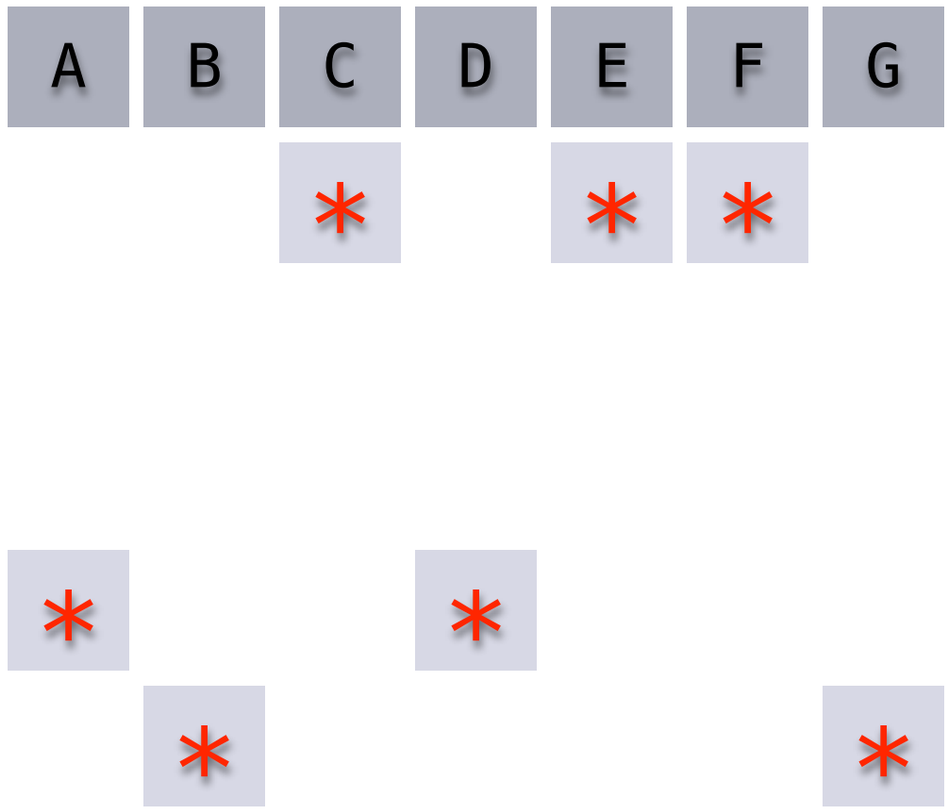
Another kind of puzzle

... find a subset of the rows ...

	A	B	C	D	E	F	G
1			*		*	*	
2	*			*			*
3		*	*			*	
4	*			*			
5		*					*
6				*	*		*

Another kind of puzzle

... with exactly one star in each column.



Another kind of puzzle

... with exactly one star in each column.

A	B	C	D	E	F	G
*	*	*	*	*	*	*

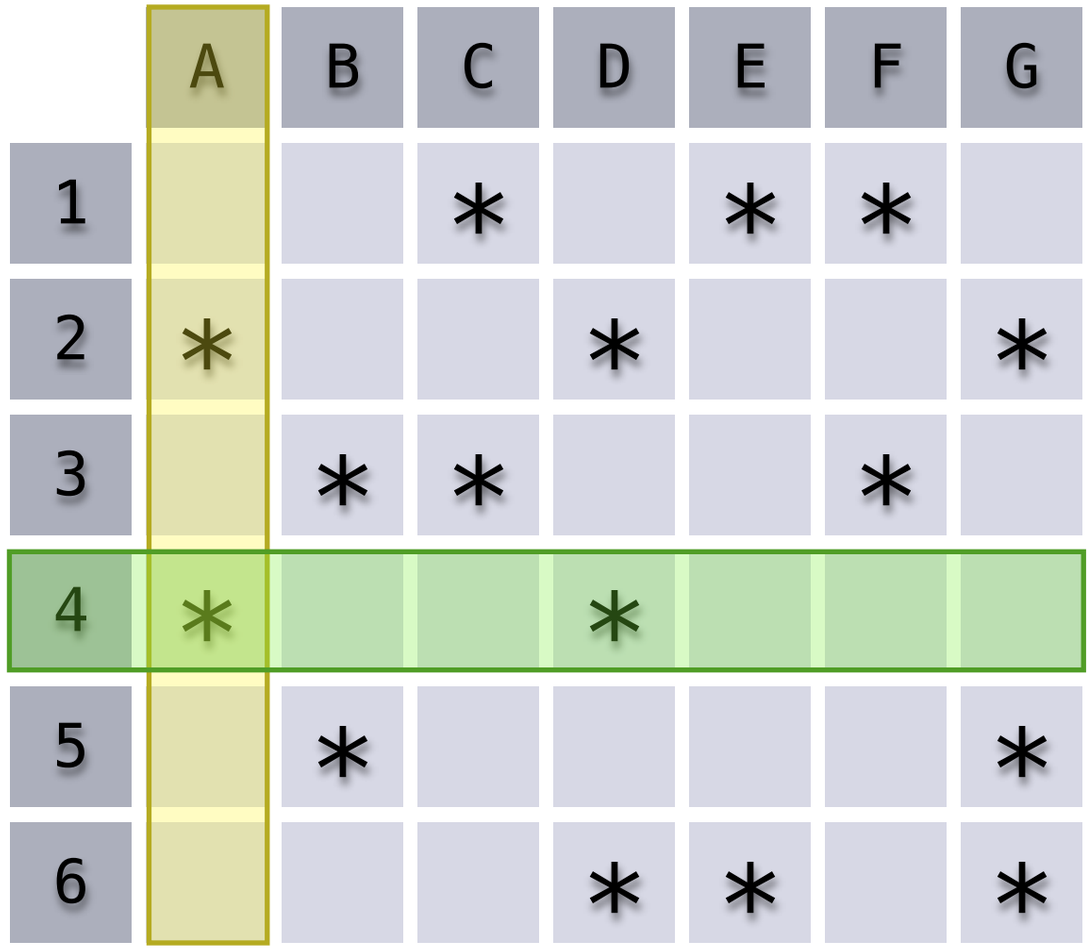
Backtracking again

	A	B	C	D	E	F	G
1			*		*	*	
2	*			*			*
3		*	*			*	
4	*			*			
5		*					*
6				*	*		*

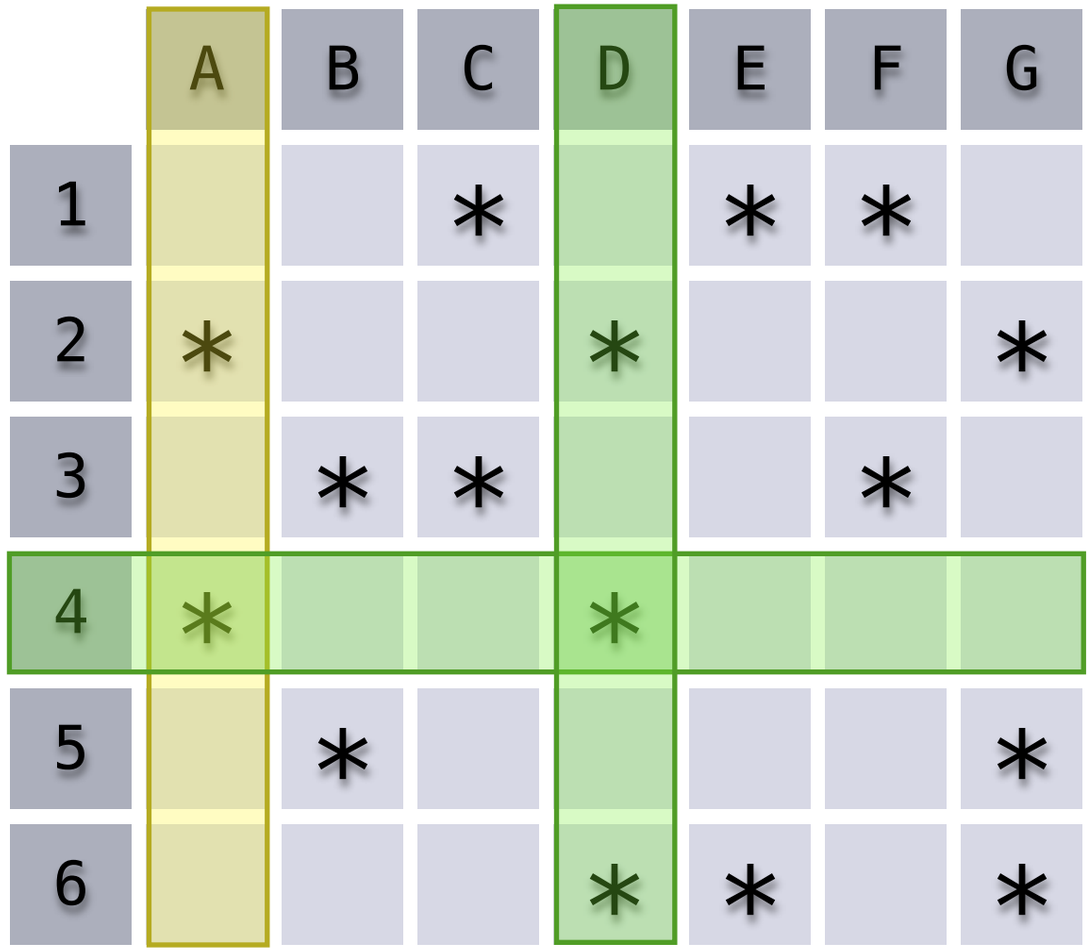
Focus on a column

	A	B	C	D	E	F	G
1			*		*	*	
2	*			*			*
3		*	*			*	
4	*			*			
5		*					*
6				*	*		*

Choose a row

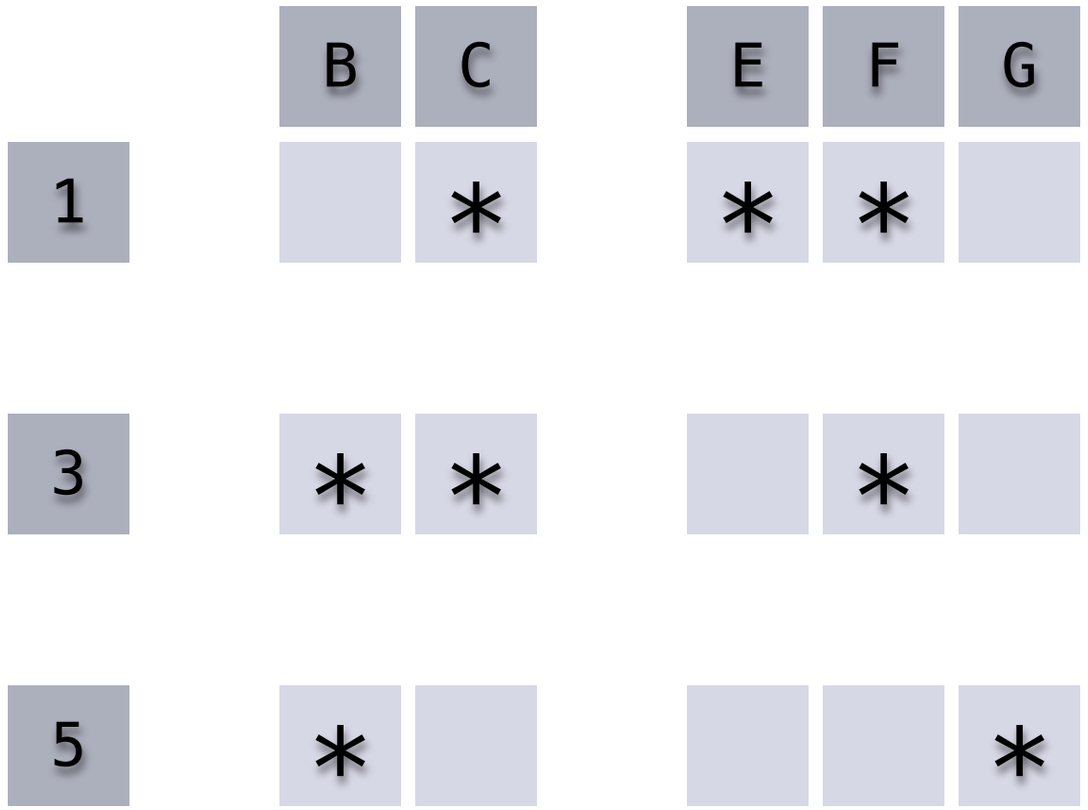


Cover other columns



Block other rows

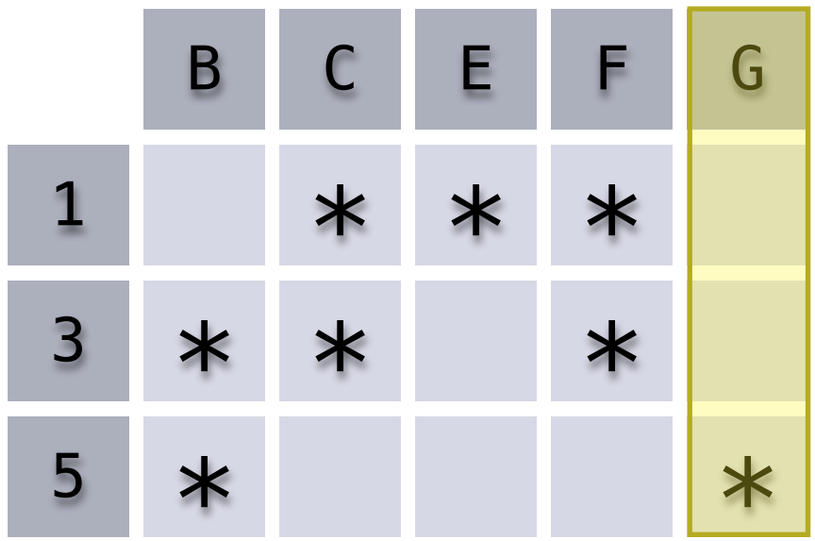
Keep what remains



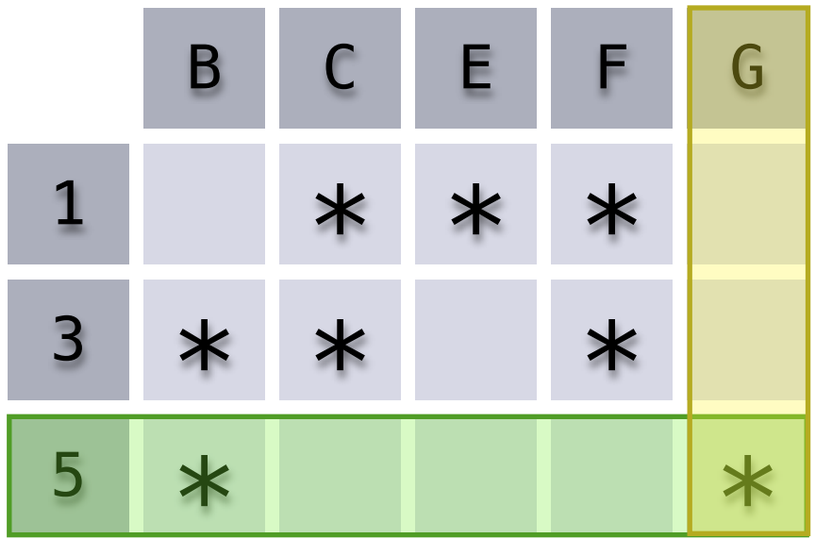
… as a problem still to solve

	B	C	E	F	G
1		*	*	*	
3	*	*		*	
5	*				*

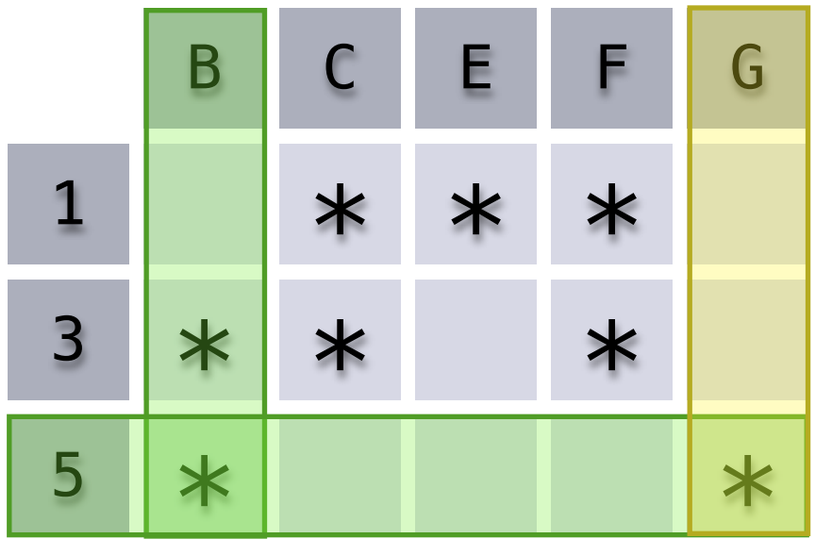
… as a problem still to solve



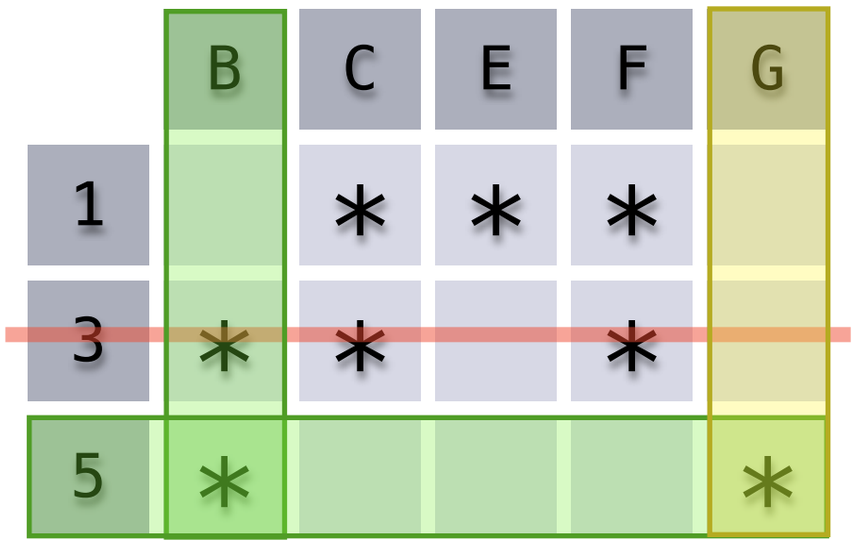
… as a problem still to solve



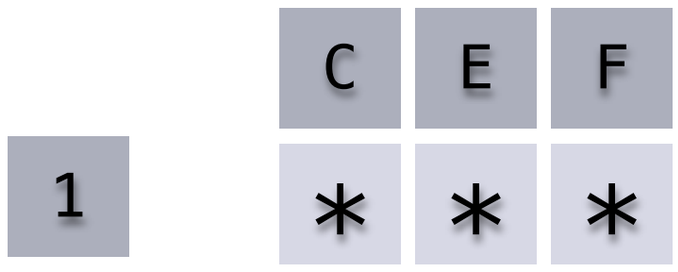
… as a problem still to solve



… as a problem still to solve



Contract again



… to form a trivial problem

C	E	F
1	*	*	*

Pentominoes

To solve pentominoes we must

- use each tile *exactly once*
- cover each square *exactly once*

… reduced to Exact Cover

So make a matrix with

- one column for each tile (12)
- one column for each square (60)

and one row for each possible move (1828).

Each move uses one tile and covers five squares:

- so each row contains 6 stars.

Interpreting the solution

- Each consistent choice of rows corresponds to a valid placement of some pieces
- A complete solution shows a solution to the pentominoes problem

So the progress of the backtracking search can be displayed on the pentomino board.

[Demo]

Smart focussing

At each stage, the search focusses on the smallest remaining column:

- Either a square that is hard to cover
- Or a piece that is hard to place.

Forced moves are taken and hopeless cases are spotted quickly.

Three programs cooperating

- *Special*: creation of the right Boolean matrix (geometry)
- *General*: solver for Exact Cover problems (backtracking)
- *Special*: display of solutions (graphics)

What about Sudoku?

Put *exactly one* digit in each box so that each digits appears:

- *exactly once* in each column,
- *exactly once* in each row,
- *exactly once* in each 3 x 3 block.

	4		6					
5	1					9		6
		8		3				4
7		1		6	2			
	8						6	
			4	7		5		1
3				4		6		
1		2					3	8
					6		9	

What about Sudoku?

Put *exactly one* digit in each box so that each digits appears:

- *exactly once* in each column,
- *exactly once* in each row,
- *exactly once* in each 3 x 3 block.

9	**4**	7	**6**	2	1	8	5	3
5	**1**	3	7	8	4	**9**	2	**6**
6	2	**8**	9	**3**	5	1	7	**4**
7	5	**1**	8	**6**	**2**	3	4	9
4	**8**	9	1	5	3	7	**6**	2
2	3	6	**4**	**7**	9	**5**	8	**1**
3	9	5	2	**4**	8	**6**	1	7
1	6	**2**	5	9	7	4	**3**	**8**
8	7	4	3	1	**6**	2	**9**	5

Sudoku and Exact Cover

Make a matrix with:

- one column for each cell (81),
- one column for each row and each digit (81),
- one column for each column and each digit (81),
- one column for each 3 x 3 block and each digit (81);

and one row for each possible move (729).

Three programs again

- Make the matrix and preselect some rows
- Find an exact cover by backtracking
- Interpret the solution as a Sudoku grid

Much easier than pentominoes, because most moves are forced.

[Demo]

Computer Science

- Not just about programming
- Nor about learning new languages
- But also about studying relationships between problems.